AIPS Memo 39

Shareable images for $\mathcal{AIPS}$ under VMS
Some Changes made to 15APR96 AIPS
(Typeset version of January 1986 Runoff Document)

Pat Moore, Gary Fickling*

May 26, 1994

Abstract

This memo describes a proposed implementation of $\mathcal{AIPS}$ using shareable images. Included in this proposal are some changes to the $\mathcal{AIPS}$ directory structure which have already been implemented in the `15APR86` release. These changes not only facilitate building shareable images under *VMS* and other systems, but tidy up the directory structure for all $\mathcal{AIPS}$ environments. Shareable images have numerous advantages, among which are that they conserve disk space and physical memory, they may be maintained and installed more easily, and they allow support for multiple devices (ie different types of TV display systems on the same computer). The new directory structure provides some of these advantages already. The second part of this memo describes the details of the current implementation that must be understood in order to program in $\mathcal{AIPS}$.

1 Introduction

Shareable images are a mechanism in `VMS` whereby executable programs are linked in such a way as to call routines that reside in other so-called shareable images. This mechanism is used extensively by `VMS` itself and has numerous advantages. Some of the more obvious ones are as follows.

- Conserve disk space — executable images are smaller
- Conserve physical memory — this can be shared between processes
- Easy maintenance — routines can be modified without having to relink all application programs
- Easy installation — AIPS could always be shipped pre-linked to `VMS` sites
- Support for multiple devices - by simply replacing a shareable image we can switch between AP and TV devices

Clearly most of the ideas in this memo refer specifically to $\mathcal{AIPS}$ under `VMS`, but there are several aspects that are significant to $\mathcal{AIPS}$ in general. The major one of these is the AIPS directory hierarchy. When building shareable images it is essential to have a clear picture of the subroutine hierarchy. This is particularly important when we want to support a variety of different AP's and TV's. It is an ideal opportunity to tidy up the directory structure.

*Both formerly at the National Radio Astronomy Observatory

2 Directory Structure

2.1 Design Guidelines

The following are some of the guidelines used in devising this scheme.

- Separate source code from all other system specific files. This source code directory tree should contain no system specific object libraries, command procedures etc., as these may well be implemented differently on different machines.
- The source code areas should be clearly organized into true standard AIPS areas and particular operating system or device specific areas. It would also be convenient to allow the existence of a few generic areas for routines that are not standard, but are useful in various environments.
- Clarify routine hierarchy to allow shareable images to be sensibly defined and to clearly reflect linking sequences.
- The subroutine and program hierarchy should be independent of any object libraries or shareable images used on a particular system. The source code directories may be assembled into object libraries etc. in any manner convenient for the system being used.
- We should allow the previous directory structure to be easily reproduced so that no changes are necessary on other working systems.
- Preserve non-standard areas so that we can keep track of programs which are or use non-standard code.
- Define search paths to automatically pick up the most suitable version of a routine. For example the search should begin with any device specific routine, then for a generic routine and finally a standard routine. The first one found should be used. This ensures that the most efficient is used, while providing less efficient more general ones to be available.
- Try to make the structure as logical and consistent as possible.
- Use the minimum number of directories consistent with the above. There will however need to be an increase in the number of directories.

2.2 Proposed Directory Structure

The proposed directory structure requires a hierarchical file system on the host computer. Given this restriction it should be easy to implement on various operating systems. It attempts to divide up the files along the following lines.

- Routine hierarchy - i.e. whether a routine makes use of the AP or TV.
- Routine type - whether a routine is a general library routine or specific to a single application program.
- Routine version - whether a routine is standard and works with all implementations, generic and works with some, or specific and only works with one implementation.

The proposed directory structure uses the first of the above as the primary division of source code. This division closely follows the division into shareable images for `VMS`. All source code is contained in five top level areas i.e. areas one level below the AIPS version node (e.g. `15OCT85`). These areas are labelled as follows:

- APL — general utility routines

- Q — AP routines
- Y — TV routines
- QY — AP and TV routines (at present only application programs)
- AIPS — POPS utility routines (may use TV also)

There are a few obvious omissions from this list, such as no attempt to formalize various graphics, terminal or network devices. These may also benefit from such a division, but at present *AIPS* has no suitably general model available. These may be added later.

These top level areas are each divided in an identical manner into three:

- Programs - application programs. Lower level areas are present for any device specific programs, or system specific linker instructions. A non-standard area is also provided.
- Utility routines - library subroutines that may call device specific routines, but are themselves device independent. A non-standard area is also provided.
- Device routines - library subroutines that are device specific. Various generic areas are also included.

In addition to these five source code areas, there are several other top level directory areas. All of these are now described in more detail. In this discussion only three operating system branches are shown, but more can easily be added. Some of these low level areas may be further sub-divided to allow for different flavours of UNIX for example.

2.2.1 APL

This area is for utility routines and programs that make no reference to an AP or TV device.

```
                       APL
        ______________/ | \___________
       /                |             \
    DEV                SUB             PGM
   / | \                |                \
  /  |  \               |                 \
VMS  COS  UNIX        NOTST              NOTST
                     /  |  \            /  |  \
                    /   |   \          /   |   \
                  VMS  COS  UNIX     VMS  COS  UNIX
```

The DEV branch is for the standard set of Z routines. Several of these have now been made generic for some operating systems, and these belong in the DEV area itself. The lower levels are for true system specific versions. The SUB branch is for routines that are in principle system independent. There is a NOTST area for those which while not fully following AIPS coding standards stand a good chance of working on many systems. The system specific areas on this branch are for peculiar non-standard routines that are not part of standard AIPS. The PGM branch is for task programs. It too has non-standard and system specific areas. Note that the system specific areas may be used to store auxilliary files needed to link programs on a particular system.

2.2.2 Q

This area is for routines and programs that make use of the AP.

```
                                                 Q
                       _________________________/|\___
                      /                          |    \
                   DEV                          SUB    PGM
                __/   \_________                 |        \
               /                \                |         \
            FPS                  PSAP          NOTST       NOTST
           /   \            _______|_______                /  |  \
          /     \          /     |     |   \              /   |   \
       32B       16B     VMS    COS   CVEX  ALLN        VMS  COS  UNIX
       /         /  \
      /         /    \
   190      120B     5000
```

The DEV branch is for the various versions of the Q routines. The DEV area itself is for the most general version of these i.e. the PSAP code. The lower level branches support a variety of different AP devices, in some cases with generic areas. Note that because of the search path mechanism these low level areas need not contain a full set of Q routines, generic ones from higher up the tree can be substituted. The SUB branch is for routines which make use of the Q routines but are themselves device independent. This includes a non-standard area, but no system specific ones. The PGM branch is for tasks which use the AP. The system specific areas in this case are only for auxilliary linking files.

2.2.3 Y

This area is for routines and programs that make use of the TV.

```
                                    Y
                        ____________/|\__
                       /             |   \
                    DEV             SUB   PGM
                 __/ | \__           |       \
                /    |    \          |        \
             IIS    DEA    STUB    NOTST      NOTST
            /   \                            /  |  \
           /     \                          /   |   \
        M70       M75                    VMS   COS   UNIX
```

This tree is very similar to the Q tree. The only difference is in the device specific DEV branch. The generic DEV area is for Y routines that really are implemented in device independent ways. These are defined as the level 0 Y routines in the Going AIPS manual. Note that there is a difference here between the Q and Y trees - all systems have some kind of AP, while some systems do not have a TV. We therefore need to be able to distinguish generic routines from stubbed routines substituted when no TV is present. This is the purpose of the STUB area.

2.2.4 QY

This area is for routines and programs that make use of the both the AP and TV. At present this only occurs at the program level so this tree is very simple.

```
 QY
   \
    \
     PGM
```

```
         \
          \
          NOTST
         /  |  \
        /   |   \
     VMS   COS   UNIX
```

2.2.5 AIPS

This area is for the POPS level programs and related routines. Several of these make use of the TV device, but as they are routines not accessible to tasks they reside here.

```
                    AIPS
              ______/ |  \______
             /        |         \
          DEV        SUB         PGM
         / | \                      \
        /  |  \                      \
     VMS  COS  UNIX                  NOTST
                                    /  |  \
                                   /   |   \
                                VMS   COS   UNIX
```

This tree is very similar to the APL tree. The only difference is that at present there are no non-standard subroutines.

2.2.6 INCLUDE

This area is for the various include files needed by routines in all the above trees.

```
                  INC
                   |
                   |
                 NOTST
           _____/  |  \_____
          /        |        \
       VMS        COS        UNIX
```

The system specific areas allow array sizes to change between systems, and also permit system specific options such as dependency directives needed by vectorizing compilers.

2.2.7 HELP

The HELP tree is very simple as all help files are in a standard format. This tree consists of a single area.

2.2.8 LOAD

This area is for load modules i.e. fully linked programs in a form ready to be run. This is split into a standard LOAD area and a few alternative areas immediately below (e.g. LOAD.ALT1). These alternate areas could, for example, be used to keep pseudo AP versions of programs.

2.2.9 LIBRARY

This area (LIBR) is for the various subroutine libraries used to build AIPS programs. Note that these have been moved out of the system independent source code areas. We may in the future wish to include several libraries not of AIPS origin along with AIPS. These would enable AIPS programs to make use of some useful code that is available in the public domain. Such libraries will be included in this area.

2.2.10 DOCUMENTATION

This area (DOC) will be identical to the existing documentation area.

2.2.11 SYSTEM

This area is used to store the various system specific tools needed by AIPS. The structure of this area is system specific.

2.3 MNEMONICS

Programmers always refer to the AIPS directory areas by means of mnemonics. These need to be implemented on various operating systems and it is convenient to store a list of them, complete with their associated areas in a file which can be used by any operating system. Below is a copy of this file. It can be used to assign the appropriate mnemonics, or to create a complete directory tree. The operating systems referred to here are not intended to be a complete list.

```
! AREAS - This file contains a complete list of all AIPS directory tree
! areas. It only includes those areas below the version directory node.
! It also includes the mnemonics used to identify areas.
! This file should be largely system independent if the following
! special characters can be dealt with:
!
!       ! comment character
!       . directory delimiter
!
! Declare the basic source code areas
!
APL             APL
Y               Y
Q               Q
QY              QY
AIP             AIPS
!
! APL subroutine areas - nothing here references Q or Y
!
APLGEN          APL.DEV                         ! generic Z
APLVMS          APL.DEV.VMS                     ! VMS Z
APLMC4          APL.DEV.MC4                     ! Modcomp Z
APLCOS          APL.DEV.COS                     ! COS Z
APLUNIX         APL.DEV.UNIX                    ! UNIX Z
APLBELL         APL.DEV.UNIX.BELL               ! Bell Z
APLSYS3         APL.DEV.UNIX.BELL.SYS3          ! Bell System 3
APLSYS5         APL.DEV.UNIX.BELL.SYS5          ! Bell System 5
APLV7           APL.DEV.UNIX.BELL.V7            ! Bell Version 7
APLUTS          APL.DEV.UNIX.BELL.V7.UTS        ! Amdahl UTS
```

```
APLMSC          APL.DEV.UNIX.BELL.SYS3.MASC ! Masscomp
APLBERK         APL.DEV.UNIX.BERK           ! Berkley
APL4PT1         APL.DEV.UNIX.BERK.4PT1      ! Berkley 4.1
APL1VAX         APL.DEV.UNIX.BERK.4PT1.VAX  ! Berkley 4.1 VAX
APL4PT2         APL.DEV.UNIX.BERK.4PT2      ! Berkley 4.2
APLVEX          APL.DEV.UNIX.BERK.4PT2.CVEX ! Berkley 4.2 Convex
APLALN          APL.DEV.UNIX.BERK.4PT2.ALLN ! Berkley 4.2 Alliant
APL2VAX         APL.DEV.UNIX.BERK.4PT2.VAX  ! Berkley 4.2 VAX
!
APLSUB          APL.SUB                 ! standard routines
APLNOT          APL.SUB.NOTST           ! non-standard routines
APLNVMS         APL.SUB.NOTST.VMS       ! VMS non-standard routines
APLNUNIX        APL.SUB.NOTST.UNIX      ! UNIX non-standard routines
APLNMC4         APL.SUB.NOTST.MC4       ! Modcomp non-standard routines
APLNCOS         APL.SUB.NOTST.COS       ! COS non-standard routines
!
! APL program areas - these reference only APL routines
!
APLPGM          APL.PGM                 ! standard programs
APGNOT          APL.PGM.NOTST           ! non-standard programs
APGVMS          APL.PGM.NOTST.VMS       ! VMS programs
APGUNIX         APL.PGM.NOTST.UNIX      ! UNIX programs
APGMC4          APL.PGM.NOTST.MC4       ! Modcomp programs
APGCOS          APL.PGM.NOTST.COS       ! COS programs
!
! Q subroutine areas
!
QDEV            Q.DEV                   ! pseudo AP Q
QFPS            Q.DEV.FPS               ! generic FPS Q
QFPS16          Q.DEV.FPS.16B           ! 16 bit FPS Q
Q120B           Q.DEV.FPS.16B.120B      ! AP120B Q
Q5000           Q.DEV.FPS.16B.5000      ! FPS 5105, 5205 ...
QFPS32          Q.DEV.FPS.32B           ! 32 bit FPS Q
Q120B           Q.DEV.FPS.32B.190       ! AP190 Q
QPSAP           Q.DEV.PSAP              ! Generic PSAP
QVMS            Q.DEV.PSAP.VMS          ! VMS specific
QCOS            Q.DEV.PSAP.COS          ! Cray-COS Q
QVEX            Q.DEV.PSAP.CVEX         ! Convex Q
QALN            Q.DEV.PSAP.ALLN         ! Alliant Q
!
! Routines that use Q routines
!
QSUB            Q.SUB                   ! routines that call Q
QNOT            Q.SUB.NOTST             ! non-standard routines
!
! Q program areas - these reference APL and Q routines
!
QPGM            Q.PGM                   ! standard programs that call Q
QPGNOT          Q.PGM.NOTST             ! non-standard programs
QPGVMS          Q.PGM.NOTST.VMS         ! VMS programs
QPGUNIX         Q.PGM.NOTST.UNIX        ! UNIX programs
QPGMC4          Q.PGM.NOTST.MC4         ! Modcomp programs
QPGCOS          Q.PGM.NOTST.COS         ! COS programs
!
! Y subroutine areas
!
```

```
YGEN           Y.DEV                  ! generic Y
YSTUB          Y.DEV.STUB             ! stubbed Y
YIIS           Y.DEV.IIS              ! generic IIS Y
YM70           Y.DEV.IIS.M70          ! model 70 Y
YM75           Y.DEV.IIS.M75          ! model 75 Y
YDEA           Y.DEV.DEA              ! Deanza Y
YV20           Y.DEV.V20              ! Comtel
!
! Routines that use Y routines
!
YSUB           Y.SUB                  ! standard routines that call Y
YNOT           Y.SUB.NOTST            ! non-standard routines
!
! Y program areas - these reference APL and Y routines
!
YPGM           Y.PGM                  ! standard programs
YPGNOT         Y.PGM.NOTST            ! non-standard programs
YPGVMS         Y.PGM.NOTST.VMS        ! VMS programs
YPGUNIX        Y.PGM.NOTST.UNIX       ! UNIX programs
YPGMC4         Y.PGM.NOTST.MC4        ! Modcomp programs
YPGCOS         Y.PGM.NOTST.COS        ! COS programs
!
! QY program areas - these reference APL, Q and Y routines
!
QYPGM          QY.PGM                 ! standard programs
QYPGNOT        QY.PGM.NOTST           ! non-standard programs
QYPGVMS        QY.PGM.NOTST.VMS       ! VMS programs
QYPGUNIX       QY.PGM.NOTST.UNIX      ! UNIX programs
QYPGMC4        QY.PGM.NOTST.MC4       ! Modcomp programs
QYPGCOS        QY.PGM.NOTST.COS       ! COS programs
!
! AIPS routine areas
!
AIPGEN         AIPS.DEV               ! Generic Z
AIPVMS         AIPS.DEV.VMS           ! VMS Z
AIPUNIX        AIPS.DEV.UNIX          ! UNIX Z
AIPMC4         AIPS.DEV.MC4           ! Modcomp Z
AIPCOS         AIPS.DEV.COS           ! COS Z
AIPSUB         AIPS.SUB               ! standard routines
!
! AIPS program areas
!
AIPPGM         AIPS.PGM               ! standard programs
AIPNOT         AIPS.PGM.NOTST         ! non-standard programs
AIPGVMS        AIPS.PGM.NOTST.VMS     ! VMS programs
AIPGUNIX       AIPS.PGM.NOTST.UNIX    ! UNIX programs
AIPGMC4        AIPS.PGM.NOTST.MC4     ! Modcomp programs
AIPGCOS        AIPS.PGM.NOTST.COS     ! COS programs
!
! Include files - note that the INCS mnemonic is NOT set up here
! as it does not refer to a single area. A special search path needs
! to be set up to enable the correct version to be accessed.
!
INC            INC                    ! standard includes
INCNOT         INC.NOTST              ! non-standard includes
INCVMS         INC.NOTST.VMS          ! VMS includes
```

```
INCUNIX         INC.NOTST.UNIX          ! UNIX includes
INCMC4          INC.NOTST.MC4           ! Modcomp includes
INCCOS          INC.NOTST.COS           ! COS includes
INCVEX          INC.NOTST.CVEX          ! CONVEX includes
INCALN          INC.NOTST.ALLN          ! Aliant includes
!
! System dependent areas - e.g. command procedures, object libraries
!
SYSVMS          SYSTEM.VMS              ! VMS system
SYSLOCAL        SYSTEM.VMS.LOCAL        ! Local mods
SYSUNIX         SYSTEM.UNIX             ! UNIX system
SYSMC4          SYSTEM.MC4              ! Modcomp system
SYSCOS          SYSTEM.COS              ! COS system
!
! Documentation
!
DOC             DOC
DOCGRIP         DOC.GRIP
DOCPUBL         DOC.PUBL
DOCTXT          DOC.TEXT
DOCWHO          DOC.WHO
!
! Various
!
HLPFIL          HELP
HIST            HIST
LOAD            LOAD
LOAD1           LOAD.ALT1
LIBR            LIBR
```

3 VMS details

The previous section described the proposed changes that will be visible with all versions of AIPS. This section details the changes needed for the `VMS` implementation.

3.1 Object Libraries

With a new source code directory structure it is possible for AIPS to use different object library structures with different operating systems as is convenient. Below is a list of object libraries suitable for `VMS`, together with a list of areas from which they are built. Note that the object library file names have been deliberately lengthened with the LIB string. This is to prevent any name conflicts with the directory area mnemonics.

- `APLSUBLIB.OLB` from `[APL.SUB]`
- `APLNOTLIB.OLB` from `[APL.SUB.NOTST...VMS]`
- `APLVMSLIB.OLB` from `[APL.DEV...VMS]`
- `QSUBLIB.OLB` from `[Q.SUB]`
- `QNOTLIB.OLB` from `[Q.SUB.NOTST]`
- `QPSAPLIB.OLB` from `[Q.DEV]`
- `Q120BLIB.OLB` from `[Q.DEV...120B]`

- `Q5000LIB.OLB` from `[Q.DEV...5000]`
- `YSUBLIB.OLB` from `[Y.SUB]`
- `YNOTLIB.OLB` from `[Y.SUB.NOTST]`
- `YSTUBLIB.OLB` from `[Y.DEV...STUB]`
- `YM70LIB.OLB` from `[Y.DEV...M70]`
- `YM75LIB.OLB` from `[Y.DEV...M75]`
- `YDEALIB.OLB` from `[Y.DEV...DEA]`
- `AIPSUBLIB.OLB` from `[AIP.SUB]`
- `AIPVMSLIB.OLB` from `[AIP.DEV...VMS]`

These object libraries would be updated exactly as now when routines are modified, by means of a new `COMRPL` procedure. There would however be a few differences. First there are a larger number of directories. This means that programmers need to know more precisely where a routine resides. It may be possible to reduce the impact of this by setting up logical names to implement search paths to find a particular routine. However, initially I suggest we do not do this so as to help ensure the programmers are aware of which version of a routine they are modifying, and any consequences it may have. Second some routines find their way into more that one object library. This is done deliberately to simplify linking procedures while still maintaining a single copy of the ultimate source. The necessary intelligence to replace a routine in multiple libraries can easily be built into the `COMRPL` procedure.

These object libraries serve two purposes. They can be used directly by a `COMTST` procedure for programs to link with directly. This is not the normal mode of operation, but is available for testing purposes. Normally the object libraries are used to build shareable images. The programs are then linked with the shareable images using the `COMLNK` procedure. These procedures are described in detail in section 6.

3.2 Shareable Images

In order to get all the benefits of shareable images we need to carefully separate out code that refers to device specific routines. At present this is only done for the Q and Y routines, but in future we may wish to extend this list. We need a minimum of 4 shareable images at present. One for application routines that are Q and Y device independent, one each for Q and Y devices and one for routines used only by POPS level programs. Note that the Q and Y shareable images must include ALL routines that use Q and Y routines, as well as the Q and Y routines themselves. This is the major motivation behind the directory reorganization. The following is a list of all the shareable images we need at present. It includes the mnemonic for the image, its file name (ignore the peculiar file names at present) and a list of object libraries from which they are constructed.

- APL A15JUL86.EXE from APLSUBLIB,APLNOTLIB,APLVMSLIB
- PSAP Q15JUL86.EXE from QSUBLIB,QNOTLIB,QPSAPLIB
- 120B Q15JUL86.EXE from QSUBLIB,QNOTLIB,Q120BLIB
- 5000 Q15JUL86.EXE from QSUBLIB,QNOTLIB,Q5000LIB
- STUB Y15JUL86.EXE from YSUBLIB,YNOTLIB,YSTUBLIB
- M70 Y15JUL86.EXE from YSUBLIB,YNOTLIB,YM70LIB
- M75 Y15JUL86.EXE from YSUBLIB,YNOTLIB,YM75LIB

- DEA Y15JUL86.EXE from YSUBLIB,YNOTLIB,YDEALIB
- POPS P15JUL86.EXE from AIPSUBLIB,AIPVMSLIB

Basically there are 4 shareable images - APL, Q, Y and POPS. There are multiple versions of Q and Y to handle various devices. These Q and Y images each contain their own copy of some high level routines that call device specific Q or Y routines.

All shareable images will be built with a set of transfer vectors at the start of the image, as recommended by DEC. This gives two advantages. First it is possible to rebuild a shareable image, and all programs linked to it continue to run without relinking (this is how DEC implement `VMS` and FORTRAN updates without user program relinking). Second, it is possible to build several versions of the Q and Y shareable images with identical entry points to these transfer vectors. It is then possible to substitute different versions of these shareable images without relinking any application programs. We can thus completely remove the old pseudo AP load area.

We can use another trick with shareable images to simplify program linking. The four fundamental shareable images - APL, PSAP, STUB and POPS could be used to form a shareable image library. Note that such a library does NOT contain useable shareable images ! It merely contains symbol table information to enable the linker to decide which images are needed, and a mechanism for locating the real shareable image EXE files. This way it does not matter which version of Q or Y go into the shareable image library as they all have the same entry points for the list of transfer vectors. At run time appropriate logical names are set up to direct the image activator to the appropriate versions of the shareable image.

There are some things we need to beware of when using shareable images. Shareable images files are located in a peculiar manner using the default file specification string `SYS_$LIBRARY:.EXE;0`. This effectively means that logical names must be used to override the `SYS_$LIBRARY` area. This is not a problem as there are several reasons for always using logical names to point to shareable images. It also means that only the top version can ever be used.

Shareable images can be used immediately without using the `VMS` INSTALL utility. All the benefits are obtained except for sharing physical memory. To gain this final benefit care must be exercised. The INSTALL utility has no way to allow more than one version of a shareable image to be accessed, so if we require to be able to switch between AP and PSAP, or between different TV devices none of the Q or Y images may be installed. There is no problem with the APL and POPS images as there is only one version. As the INSTALL utility requires privilege to be used anyway the only sensible approach to use is to not install any images by default. Each site can if it chooses install a selection of the shareable images depending upon its requirements.

There is a problem running multiple versions of *AIPS*. At NRAO we will have three versions active at one time - OLD, NEW and TST. So there will be 3 versions of APL and POPS, 9 versions of Q and 12 versions of Y ! We must be very careful with naming conventions in view of the peculiar way shareble images are accessed by `VMS`. This is the reason for including the AIPS version in the file names listed previously. We shall be forced to use separate directories to hold different versions of shareable images with the same name.

The *AIPS* installation procedure can be dramatically simplified by the use of shareable images. All programs can now be shipped fully linked regardless of the local AP or TV device. Similarly object libraries for building the shareable images can also be supplied pre-compiled. All that needs to be done is to build the shareable images (typically 4) for the particular hardware configuration using the object libraries. This should be very quick. Alternatively it may be possible to supply the shareable images fully linked also. The installaion procedure simply selects the ones needed for each site. This latter procedure may, however, be confusing due to the peculiar naming conventions for shareable images.

The use of transfer vectors with shareable images is both useful and annoying. It is useful because it hides all internal routine names. This will help to avoid name conflicts with various alternative libraries used alongside AIPS. It also enables certain AIPS routines to be invisible to application code. The ZQ routines are an obvious example here. The problem is that the list of transfer vectors needs maintaining. I would however like to point out the similarity of maintaining transfer vectors and routine shopping lists. With suitable comment fields in the code both these tasks could be performed automatically.

Another maintenance difficulty with shareable images come from FORTRAN COMMON blocks. Common blocks that reside totally at the application program level are no trouble. There are however two problems for ones that reside within shareable images. First the FORTRAN compiler gives them the SHR attribute. This means that when shareable images are installed these common blocks are common to all processes using them ! This is not what AIPS requires. To avoid the problem the linker needs to be told the name of every common block so that it can modify the SHR attribute. We thus need to maintain a list of common blocks used by AIPS subroutines. This does not need to be kept tidy at present as the linker simply ignores attempts to change the attributes of non-existent common blocks. This task too could be automated by the addition of comment fields to the various include files to build a suitable list of linker commands. The second problem is more serious and is due to AIPS violating the FORTRAN 77 standard. This states that named common blocks must be the same length in every module. AIPS allows application programs to extend common blocks beyond the size in the subroutines. The problem with shareable images is that space for the common blocks is reserved in the shareable image itself, which is built from the subroutines alone. This abuse of common blocks needs to be carefully considered, and if possible removed. If it cannot be removed a slightly modified set of include files will need to be made to ensure sufficient space is reserved in the shareable images to accomodate all programs.

4 implementation

Shareable images provide numerous advantages for AIPS under `VMS`. We should make use of these advantages as soon as possible. The best approach is probably to make the changes step by step, to enable each one to be tested thoroughly. The critical first step is to move to the new directory structure. This has been done for the 15APR86 version of AIPS. The tools needed for this new structure have been written and are described in section 6. With the new directory structure in place we can then build and test all the tools we need to use shareable images. Finally we can switch over to start using them in the near future.

5 New File Names For Data.

In addition to the directory structure changes just outlined, another change has been made to the 15APR86 version of AIPS. The disk volume field for data files has been replaced by an "AIPS version letter". The letter we are using for 15APR86 will be "A". The next change in file data formats will require us to change the letter to "B". It should be quite sometime before we get to "Z", thus creating a crisis. As an example, the 15OCT85 format map file MA201501.221 will be renamed to MAA01501.221 in the 15APR86 release.

This change will bring a number of advantages:

- Data backed up by `VMS` BACKUP can be restored to a different disk.
- Multiple dismountable disk driver are now better supported. Previously a disk written as AIPS disk 2 had to always be mounted as AIPS disk 2.
- Out of date data will be ignored by a given release of AIPS, thus preventing many potential disasters. Out of date data can be easily detected by looking at file names.
- An intelligent data file update program (UPDAT) has been written. This can now recognise what version of input data it is being fed.

Files that are shared among users (and between different versions) such as system parameter files, accounting files, batch files, etc. are found in the directory pointed to by logical name DA00 and have a "1" in the AIPS version letter field (the "1" doesn't signify anything).

Memory files are in a version specific area [AIPS.date.MEMORY]. These files have a "1" in the AIPS version letter field.

6 A TUTORIAL FOR PROGRAMMERS ON USING THE NEW TOOLS

6.1 Initialization And Startup Procedures

6.1.1 LOGIN.PRG

The logical names and symbols needed to program in AIPS (and to run AIPS) can be obtained by executing command procedure LOGIN.PRG. A programmer should put the following line (substituting the disk used for AIPS at his site for "AIPS_DISK_NAME") in his LOGIN.COM file:

```
$ @AIPS_Disk_Name:[AIPS]LOGIN.PRG
```

At NRAO This procedure makes TST the default AIPS_VERSION. Other sites may only have one AIPS_VERSION (NEW) and may have things set up diferently.

6.1.2 AIPS 'Version' 'Option'

This procedure will startup a given version of AIPS. On CVAX 'Version' can be either OLD, NEW or TST. One can also start up AIPS with the following options:

```
REMOTE - Used to run AIPS from a TEK graphics terminal.
DEBUG  - Run AIPS with the debugger.
LOCAL  - Run a private AIPS found in the current default directory.
```

The DEBUG option works only if the standard AIPS is linked with debug or if you use the LOCAL option and you have an AIPS linked with debug in your current default directory.

6.2 Compiling and Linking.

6.2.1 COMRPL 'SubroutineSpec' 'Option'

This routine will compile and replace a subroutine or set of subroutines in the proper AIPS library. The 'Option' field, if present, MUST follow the 'Subroutine Spec' field, rather than precede it. The parameter 'SubroutineSpec' can be a single logical name and subroutine such as APLSUB:CTICS, or it can be a list of subroutines such as APLSUB:CTICS,COPY,APLNOT:CHKTAB, or it can be a wild card such as APLSUB:CH*.*, or it can be a file containing a list or routines such as @MYLIST.TXT (the "@" signifies a file). Note that to specify the directory of the subroutine, you MUST use a logical name such as APLSUB rather than the full directory specification such as [AIPS.15APR86.APL.SUB]. The procedure uses the standard AIPS defaults (NOI4, NOOPTIMIZE, DEBUG, WARNINGS=ALL) with the compile (FORTRAN) command. You may use any of the valid FORTRAN options listed at the end of this section. If you want to use more than one option then separate them with at least one blank. For example, the following command will compile subroutine CHCOPY, replace it in the standard AIPS library area, produce a listing and produce no warning messages for undeclared variables, tabs, and lower case code (DIRTY option).

```
$ COMRPL APLSUB:CHCOPY LIST DIRTY
```

The following examples show how multiple files can be compiled.

```
$ COMRPL APLSUB:MSGWRT,APLNOT:NXTFLG  ! Compile MSGWRT and NXTFLG.
$ COMRPL APLSUB:MP2*.FOR              ! Compile every routine whose
                                      ! name begins with MP2.
$ COMRPL @MYLIST.TXT                  ! Compile every routine listed
                                      ! in MYLIST.TXT
```

6.2.2 COMLNK 'ProgramSpec' 'Option'

This procedure will compile and link a program or set of programs and put them in the AIPS "LOAD" area. If any alternate areas are set up, such as the psuedo AP area, then a module linked with alternate libraries will be put in the alternate areas. The 'ProgramSpec' may be a list of programs, a wild card, or a file containing a list of programs as described in the COMRPL explanation. The 'Option' may be any of the list of options at the end of this section.

6.2.3 COMTST 'ProgramSpec' 'Option'

This is a version of COMLNK designed for compiling and linking experimental AIPS programs in a programmers own area. This procedure will compile and link a program or set of programs and put the executable module in the current default directory. This routine also uses an option file 'ProgramName'.OPT if it exists or LOCAL.OPT if it does not. At least one of these option files MUST be found in the default directory. Option files are used to specify which libraries and routines to link with a program. A programmer will usually copy the appropriate COMLNK option file to his own area for use with COMTST. COMLNK finds its option files in AIPS_PROC by following this rule: If a program is found in a directory XYZ, then its option file is AIPS_PROC:XYZOPT.OPT. If an alternate LOAD area exists for a program (such as the pseudo AP area) then COMTST uses AIPS_PROC:XYZOPT1.OPT to link the alternate executable module. A programmer working with MX (which is found in QYPGNOT) will copy AIPS_PROC:QYPGNOTOPT.OPT to his own area and rename it LOCAL.OPT or MX.OPT. If a programmer wants to use the pseudo AP libraries instead, then he will copy AIPS_PROC:QYPGNOTOPT1.OPT to his area and rename it LOCAL.OPT or MX.OPT. These option files can also be used as a means of specifying experimental subroutines or libraries. For instance, a programmer working on MX may copy AIPS_PROC:QYPGNOTOPT.OPT into MX.OPT and then put the names of any experimental subroutines or libraries in MX.OPT. A full example is given in the section "COMPILING AND LINKING, AN EXAMPLE".

6.2.4 Options

The following options can be used with the compile and link procedures:

Option	Minimum Abbreviation	Comments
DEBUG	DE	LINK with DEBUG (compile is always debug)
NODEBUG	NODE	LINK without DEBUG (Default)
LIST	LI	produce compiler listing
NOLIST	NOLI	no listing (Default)
MAP	MA	produce LINKER map.
NOMAP	NOMA	no linker map (Default)
OPTIMIZE	OP	compile optimized and NODEBUG.
NOOPTIMIZE	NOOP	compile nooptimized (Default)

```
DIRTY        DI        no warnings for undeclared variables, tabs
NODIRTY      NODI      warnings for undecl var, tabs (Default)
PURGE        PU        purge executable after link (Default)
NOPURGE      NOPU      do not purge executable
```

6.3 miscellaneous routines

6.3.1 VERSION 'Version'

This command will set the default version to 'Version'. 'Version' can be either OLD, NEW or TST. The version will stay in effect until the programmer changes it, or logs off. Note that when starting up the AIPS program this command is executed to select the version of AIPS to be used. This procedure should be used (with 'Version' NEW) before checking out programs from NEW, or compiling and linking NEW routines. To again use the TST version, use the procedure with 'Version' set to TST.

6.3.2 FORK 'command'

FORK is useful for running things, such as links and compiles, as subprocess. It is defined to be

```
SPAWN/NOWAIT/NOTIFY/INPUT=NLA0:/OUTPUT=FORK.LOG"
```

The following example shows how to compile and link IMLOD in a subprocess:

```
$ FORK COMLNK IMLOD
```

6.3.3 FLOG

This command is defined to be "TYPE FORK.LOG" and will type the latest FORK log file in the current directory.

6.4 Compiling and linking, an example.

This example shows how we can compile and link an experimental version of program MX with experimental versions of subroutines GRDAT, and DSKFFT, and keep the executable image in our own directory.

First, we set our default to some work directory and copy the current versions of MX, DSKFFT, and GRDAT from QYPGNOT, and APLNOT. Programmers on CVAX should copy the routines using the code checkout system.

Next, we need an option file to tell the linker what subroutines and libraries to use. MX is found in QYPGNOT so we copy over the option file for the QYPGNOT programs and rename it to LOCAL.OPT or MX.OPT. This can be done using the following command:

```
$ COPY AIPS_PROC:QYPGNOTOPT.OPT LOCAL.OPT
```

QYPGNOTOPT not only works for MX, but since it has every library (except for the POPS language processor stuff) in it, it can also be used to link any task with the standard AIPS subroutines.

To make our experimental version of GRDAT and DSKFFT link with MX, we can use the text editor to change LOCAL.OPT which looks like this:

```
.

LIBR:QNOTLIB/LIB,LIBR:APLNOTLIB/LIB,-
LIBR:QSUBLIB/LIB,-
LIBR:Q120BLIB/LIB,-
LIBR:YSUBLIB/LIB,LIBR:YM70LIB/LIB,-
LIBR:APLSUBLIB/LIB,LIBR:APLVMSLIB/LIB,LIBR:APLSUBLIB/LIB,-
FPS:HSRLIB/LIB,FPS:FPSLIB/LIB
```

. to make it look like this:

```
GRDAT,DSKFFT,-
LIBR:QNOTLIB/LIB,LIBR:APLNOTLIB/LIB,-
LIBR:QSUBLIB/LIB,-
LIBR:Q120BLIB/LIB,-
LIBR:YSUBLIB/LIB,LIBR:YM70LIB/LIB,-
LIBR:APLSUBLIB/LIB,LIBR:APLVMSLIB/LIB,LIBR:APLSUBLIB/LIB,-
FPS:HSRLIB/LIB,FPS:FPSLIB/LIB
```

The "-" is the line continuation indicator in option files.

Now we make the changes to GRDAT, DSKFFT and MX. Then we compile and link them with the following commands (the DEBUG on the COMTST command is optional):

```
$ FORTRAN/NOI4/DEBUG/NOOPTI GRDAT
$ FORTRAN/NOI4/DEBUG/NOOPTI DSKFFT
$ COMTST MX DEBUG
```

Suppose we want to link MX with debug and have the link run as a subprocess. Then we can type in

```
$ FORK COMTST MX DEBUG
```

We will be notified when COMTST finishes (or aborts!). We should type FORK.LOG (we can use the FLOG command) to make sure our task compiled and linked correctly.

6.5 Check out system.

Programmers at NRAO must use the checkout procedure to change AIPS code. All directories should be specified using the logical names instead of the full directory names. The programmer must make sure that AIPS_VERSION is set correctly. AIPS_VERSION will be TST after a programmer executes LOGIN.PRG, but AIPS_VERSION can be set to NEW if the programmer runs the NEW version of AIPS or sets the version to NEW using the VERSION command.

To check things out of NEW, the programmer should use the command

```
$ VERSION NEW
```

to set the programmer's current working version to NEW. The version can be reset to TST with the command

```
$ VERSION TST
```

A task that is still checked out of NEW can not be checked out of TST or vice versa.